UNIVERSIDAD NACIONAL AUTÓNOMA DE MÉXICO.

FACULTAD DE INGENIERÍA.

Estructura y Programación de Computadoras.

M. I. Alberto Navarrete Hernández.

Traductor de ensamblador a binario.

Integrantes:

- Edgar Alan Ramírez Martín.
- Baltazar Vallin Carlos Ulises.
- García Chávez Gael Alonso.

Semestre: 2023-2

Grupo: 03

Fecha de entrega:19 de Junio del 2023

Una vez analizados los requerimientos, se tienen 5 grupos de instrucciones diferentes los correspondientes a operaciones, los de manejo de memoria, el de carga, salto condicional y salto incondicional. El total de las operaciones es de 16 ya que 10 corresponden al primer grupo, 2 al segundo, 1 al tercero, 1 al cuarto y por último 2 al quinto, **esto lleva a la conclusión de que la tabla de operaciones debe representar 4 bits en código máquina.**

Ahora analizando el formato de operación aritmética se tiene lo siguiente: el nombre de la operación y tres direcciones más, por lo que se puede pensar que si cada dirección es de 4 bits y la operación también es de 4 bits, **una posible instrucción general de programa sería tener 4 campos el primero para el código de operación (4 bits) y 3 campos también de 4 bits para completar los 16 bits.**

Para cada grupo de operaciones la instrucción general varía, por lo que habrá que ver la posibilidad de adaptar dichas instrucciones a la propuesta anterior.

En el caso de la instrucciones aritmético-lógicas se puede ya que los tres campos restantes serían direcciones, esto lleva a una nueva conclusión; **el tamaño de los registros de propósito general es de 4 bits, con el fin de incluir la referencia de cada registro en el código binario,** de está manera se tendrá una tabla de registros que contará con 16 de estos tomando en cuenta que los últimos dos registros son para la lectura y escritura en memoria.

Para las instrucciones aritmético-lógicas la instrucción general es la siguiente:

op_code(4 bits)	dir_destino(4 bits)	dir_op_1(4 bits)	dir_op_2(4 bits)

Las instrucciones de manejo de memoria tienen la instrucción y dos direcciones de memoria, como las direcciones de memoria son de 4 bits entonces solo faltaría llenar con ceros el último campo, por lo que ésta operación se adapta a la propuesta.

En este queda como sigue:

op_code(4 bits)	dir_1(4 bits)	dir_2(4 bits)	0000

Para el salto incondicional de la operación son 4 bits y una etiqueta de 8 bits por lo que igual llenando con ceros el último campo también queda.

op_code(4 bits)	dirección de la etiqueta (8 bits)	0000

En el caso de cargar un valor inmediato se tiene la instrucción (op_code), una dirección de 4 bits y el valor inmediato de 8 bits, que quedan exactas.

op_code(4 bits)	dir_destino(4 bits)	número decimal (8 bits)

Y por último para los saltos condicionales se tiene la operación y una dirección suman 8 bits y la etiqueta son otros 8 bits que suman los 16 bits requeridos.

op_code(4 bits)	dir_1(4 bits)	dirección de la etiqueta (8 bits)

En el caso de las etiquetas su referencia puede ser adelantada y para estos casos hay que tomar en cuenta otra tabla de símbolos que almacenará el nombre de la etiqueta y la dirección, la cuál se transformará en un número binario de 8 bits.

Hay un par de detalles que restan por analizar; en la utilización de cargar un valor se decidió cargar dicho valor en los registros, la razón de esto es que no hay una instrucción para mover el valor de un registro a otro y si se ponen las direcciones de las variables en memoria se pueden confundir con las referencias a los registros. Por otro lado está el uso de las variables en memoria ya que deben ser declaradas al inicio del programa con el fin de que el programador conozca sus direcciones y de puedan almacenar en los registros $rre y $rrw, quedará más claro en ejemplo práctico.

Ya con todos estos elementos se satisfacen los requerimientos solicitados.

Las tablas utilizadas quedan como a continuación se muestra:

op_code	**código binario**
add	0000
sub	0001
mul	0010
div	0011
mod	0100
root	0101
cmp	0110
and	0111
or	1000
xor	1001
read	1010
write	1011
jmp	1100
jg	1101

jl	1110
load	1111

La tabla de registros es la siguiente:

nombre del registro	**código binario**
$r0	0000
$r1	0001
$r2	0010
$r3	0011
$r4	0100
$r5	0101
$r6	0110
$r7	0111
$r8	1000
$r9	1001
$r10	1010
$r11	1011
$r12	1100
$r13	1101
$rre	1110
$rwr	1111

Los últimos dos registros son utilizados en las operaciones read y write.

La manera de realizar el proceso de traducción se divide en dos pasadas.

Primera pasada.
La primera pasada realiza un análisis léxico de la cadena, aquí se determina si la entrada es una posible operación, un símbolo o un error, en el caso que sea una instrucción como se describió anteriormente se debe sustituir la cadena por su correspondiente valor en binario, en el caso del análisis de los registros es lo mismo, lo único que queda pues es el manejo de símbolos en el caso de las variables estas deberán ser declaradas al inicio y con las etiquetas solo se pondrá el nombre de la etiqueta en el código intermedio, si se declara

una etiqueta o variable estás deberán ser guardadas en la tabla de símbolos con su respectiva dirección de programa, las direcciones se asignan empezando en cero, incrementando en 1 y tomando en cuenta que si hay etiquetas más allá de la dirección 255 no se podrá colocar su dirección por el tamaño. La salida de está etapa es un código intermedio con el número y los nombres de las etiquetas.

Segunda pasada.
Se toma el código intermedio y se agregan las direcciones de las etiquetas en formato binario de ser posible, en el caso del salto incondicional se llenan con ceros los últimos 4 bits y con esto ya se tiene el código completo.

En el caso de la sintaxis es muy similar a la utilizada en NASM y a continuación se muestra el ejemplo para que se pueda apreciar la misma.

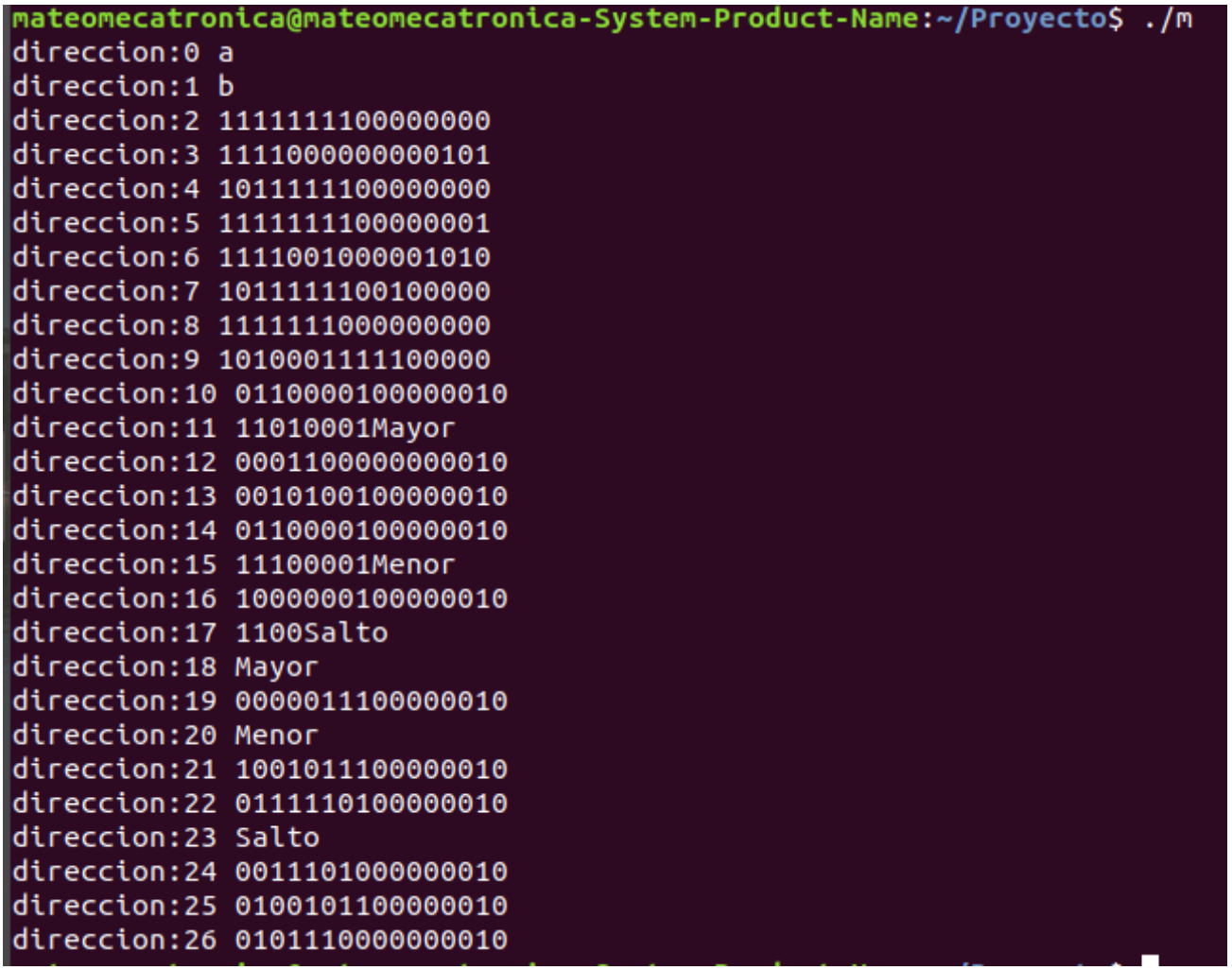
```
a;
b;

load $rwr, 0
load $r0, 5
write $rwr, $r0

load $rwr, 1
load $r2, 10
write $rwr, $r2

load $rre, 0
read $r3, $rre

cmp $r1, $r0, $r2
jg $r1, Mayor
sub $r8, $r0, $r2
mul $r9, $r0, $r2
cmp $r1, $r0, $r2
jl $r1, Menor
or $r1, $r0, $r2
jmp Salto

Mayor:
        add $r7, $r0, $r2

Menor:
        xor $r7, $r0, $r2
        and $r13, $r0, $r2
Salto:
        div $r10, $r0, $r2
        mod $r11, $r0, $r2
        root $r12, $r0, $r2
```

Archivo_fuente.txt

Descripción del programa.

Las primeras líneas declaran dos variable de nombre a y b, las instrucciones siguientes ya se describieron anteriormente, solo hay que tener en cuenta que para utilizar un salto condicional este deberá ser precedido de cmp y en la instrucción de salto condicional en nuestro jg (mayor que) o jl (menor que) en el primer registro se deberá poner el resultado de dicha operación pues no estamos tomando en cuenta un registro de banderas.

Después de ejecutar el programa el resultado es el siguiente:



En la terminal se muestra el código intermedio con sus respectivas direcciones de programa, el código intermedio también se genera en un archivo de texto pero este no contiene las direcciones aquí mostradas. Solo se muestran las direcciones con el fin de corroborar que la traducción se está llevando a cabo de forma correcta.

Finalmente a continuación se muestran los archivos .txt generados por el programa.

```
1111111100000000
1111000000000101
1011111100000000
1111111100000001
1111001000001010
1011111100100000
1111111000000000
1010001111100000
0110000100000010
11010001Mayor
0001100000000010
0010100100000010
0110000100000010
11100001Menor
1000000100000010
1100Salto
0000011100000010
1001011100000010
0111110100000010
0011101000000010
0100101100000010
0101110000000010
```

Código_intermedio.txt

```
1111111100000000
1111000000000101
1011111100000000
1111111100000001
1111001000001010
1011111100100000
1111111000000000
1010001111100000
0110000100000010
1101000100010011
0001100000000010
0010100100000010
0110000100000010
1110000100010101
1000000100000010
1100000110000000
0000011100000010
1001011100000010
0111110100000010
0011101000000010
0100101100000010
0101110000000010
```

Código_binario.txt